Biologically Inspired Computing

EECS 6180

Homework 2

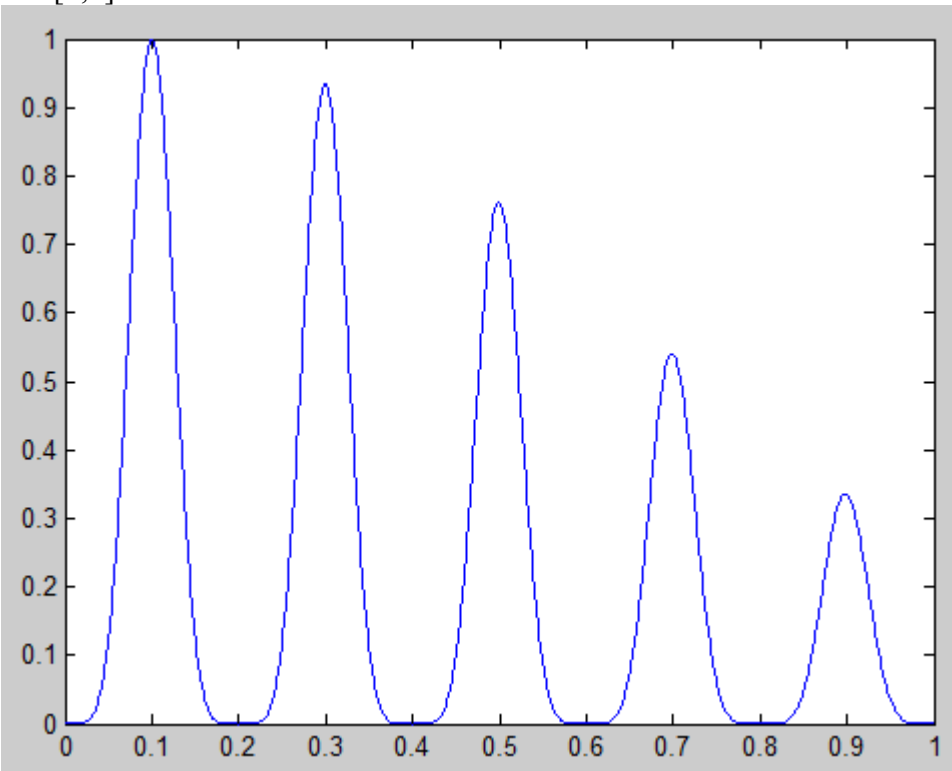
Hill Climbing

Edris Amin

Feb. 09, 2011

$F(x) = 2^{-2((x-0.1)/0.9)2} (\sin(5\pi x))^6$

$X \varepsilon [0,1]$



1. Iterated Hill Climbing

First X	X Result	#Guesses
0.701	0.701	1
0.327	0.096	6
0.686	0.105	11
0.966	0.300	16
0.946	0.097	21
0.405	0.100	26
0.660	0.100	31
0.737	0.100	36
0.215	0.101	41
0.236	0.100	46
0.100	0.100	51
0.814	0.100	56
0.416	0.101	61
0.752	0.100	66
0.599	0.101	71
0.924	0.101	76
0.489	0.102	81
0.957	0.100	86
0.468	0.102	91
0.624	0.099	96

The iterated Hill Climb algorithm seemed to converge much faster than the simple Hill Climb. As the number of guesses was increased the Iterated Hill climb algorithm converged more accurately. This method was sensitive to the # of guesses as well as the maximum number of iterations allowed to run in the embedded Simple Hill Climb was set very low to only 5. The Iterated Hill Climb is much more suited for finding Global Maxima.

2. Simple Hill Climbing

First X	X Result	iterations
0.253	0.300	500
0.563	0.497	500
0.389	0.496	500
0.472	0.506	500
0.687	0.703	500
0.881	0.901	500
0.572	0.498	500
0.428	0.484	500
0.270	0.280	500
0.685	0.710	500
0.987	0.894	500

For the simple Hill Climbing, different initial guesses of X were made to compare with results from different starting points. This showed that the Simple Hill Climbing technique was sensitive to the initial guess. The table shows that the results were usually local maxima near the initial guess. The perturbation factor may have some effect on the result allowing for ‘x’ to perturb under another likely better global guess. The Simple Hill Climb Algorithm is dependable for finding local maxima.

3. Stochastic Hill Climbing

First X	X Result	iterations	initT
0.824	0.824	1	0.100000
0.550	0.300	1	0.004762
0.399	0.503	2	0.002439
0.921	0.312	3	0.001639
0.871	0.309	4	0.001235
0.814	0.502	5	0.000990
0.824	0.098	6	0.000826
0.219	0.095	7	0.000709
0.379	0.094	8	0.000621
0.486	0.109	9	0.000552
0.222	0.102	101	0.000498
0.449	0.103	111	0.000452
0.945	0.299	121	0.000415
0.757	0.100	131	0.000383
0.017	0.100	141	0.000356
0.826	0.101	151	0.000332
0.301	0.100	161	0.000312
0.114	0.101	171	0.000293
0.930	0.100	181	0.000277
0.222	0.099	191	0.000262
0.387	0.099	201	0.000249
0.648	0.100	211	0.000238
0.458	0.101	221	0.000227
0.890	0.100	231	0.000217
0.949	0.100	241	0.000208
0.438	0.098	251	0.000200
0.327	0.101	261	0.000192
0.755	0.101	271	0.000185
0.248	0.100	281	0.000178
0.615	0.101	291	0.000172
0.759	0.100	301	0.000166
0.581	0.100	311	0.000161
0.251	0.099	321	0.000156
0.848	0.098	331	0.000151
0.734	0.100	341	0.000147
0.366	0.100	351	0.000143
0.622	0.100	361	0.000139
0.981	0.100	371	0.000135
0.180	0.100	381	0.000131
0.344	0.100	391	0.000128
0.917	0.100	401	0.000125
0.199	0.100	411	0.000122
0.253	0.100	421	0.000119
0.713	0.100	431	0.000116
0.568	0.100	441	0.000114
0.087	0.100	451	0.000111
0.772	0.100	461	0.000109
0.801	0.100	471	0.000106
0.649	0.100	481	0.000104
0.841	0.100	491	0.000102

The Stochastic Hill Climb seems more reliable to finding global maxima than the Simple Hill climb. The code was run to analyze the sensitivity to the value of ‘T’ in the the fitness determination function

$P = (1/(1+\exp((y-y1)/T)))$

As the value of T decreased the value of X converged much faster to ‘0.1’.

4. Simulated Annealing Hill Climbing

First X	X Result	iterations	Degradation
0.396	0.373	2	0.000
0.307	0.297	5	0.020
0.308	0.296	5	0.040
0.414	0.502	6	0.060
0.441	0.493	6	0.080
0.862	0.897	7	0.100
0.157	0.089	7	0.120
0.921	0.891	8	0.140
0.957	0.893	8	0.160
0.408	0.490	9	0.180
0.205	0.295	9	0.200
0.215	0.298	10	0.220
0.218	0.287	10	0.240
0.521	0.498	11	0.260
0.513	0.500	11	0.280
0.291	0.298	12	0.300
0.880	0.885	12	0.320
0.927	0.892	13	0.340
0.368	0.294	14	0.360
0.947	0.890	14	0.380
0.755	0.693	15	0.400
0.210	0.297	16	0.420
0.792	0.698	17	0.440
0.610	0.692	18	0.460
0.913	0.888	18	0.480
0.620	0.699	19	0.500
0.404	0.496	21	0.520
0.611	0.697	22	0.540
0.526	0.491	23	0.560
0.181	0.097	24	0.580
0.118	0.095	26	0.600
0.089	0.102	28	0.620
0.378	0.300	29	0.640
0.442	0.493	31	0.660
0.338	0.296	34	0.680
0.086	0.096	36	0.700
0.586	0.503	39	0.720
0.481	0.496	43	0.740
0.956	0.899	47	0.760
0.744	0.698	52	0.780
0.679	0.698	57	0.800
0.735	0.696	64	0.820
0.828	0.902	73	0.840
0.111	0.099	84	0.860
0.737	0.697	99	0.880
0.678	0.700	119	0.900
0.217	0.300	151	0.920
0.000	0.101	202	0.940
0.557	0.491	306	0.960
0.879	0.899	617	0.980

The Simulated Annealing Hill Climb algorithm seemed to act like the Simple Hill climb in that it was good at finding local maxima. However, the maxima it found were not as accurate as those found by the Simple Hill Climb, this seems to depend a bit on the

Degradation factor (G). The degradation factor was applied to T in that T = G*T, reducing the next value of T. This was done from analyzing the sensitivity of Stochastic Hill Climb. To truly find the variables that Simulated Annealing Hill climb is sensitive to further testing is required with different functions 'P' because the function

P= (-abs(exp(y-y1)) / T)

Conclusion

In all cases the number of iterations showed an improved guess of finding better maxima (local and global).

For finding Global Maxima the following things were important:

Perturbation:

X` = X +/- 1/1000 (1)
X` = X +/- 1/100 (2)
X` = X +/- 1/10 (3)

If the perturbation was very close to x like (1) this would make it likely to converge to a local maxima. Picking a perturbation function between (2) and (3) allows for a better chance of finding a global of better local maxima.

'T' value in P functions:

The T value in the Stochastic and Annealing algorithms seem to act like a variance control on the next accuracy of the guess. Therefore random process methods should be applied at finding an optimum T value for different data sets.

Master Generator:

```
%Edris Amin
%Bio Inspired Hill Climbing Homework

x=0:0.001:1;
i = size(x,2);
y=zeros(1,i);
for p=1:i
y(1,p) = 2^(-2*(((x(1,p)-0.1)/0.9)^2))*(sin(5*pi*x(1,p)))^6;
end
plot(x,y)

maxit = 500;
goal = 0.1;


disp('Simple Hill Climbing');
info = sprintf('First X\tX Result\titerations'); %Common Column Headers
disp(info);
timetotal = 0;
for i = 0:0.1:1
    [x, xinit, iter] = HILLCLIMBsimple(maxit, goal);
    row = sprintf('%1.3f\t%1.3f\t\t%i',xinit, x, iter);
    disp(row);
end

disp('|')
disp('|')

disp('Iterated Hill Climbing');
info = sprintf('First X\tX Result\t#Guesses'); %Common Column Headers
disp(info);
timetotal = 0;
for i = 1:5:100
    [x, xinit, iter] = HILLCLIMBiterated(i, 5, goal);
    row = sprintf('%1.3f\t%1.3f\t\t%i\t',xinit, x, i);
    disp(row);
end

disp('|')
disp('|')

disp('Stochastic Hill Climbing');
info = sprintf('First X\tX Result\titerations\tinitT'); %Common Column
Headers
disp(info);
for i = 1:20:1000
    [x, xinit, iter, initT] = HILLCLIMBstochastic(i, i/2, goal);
    row = sprintf('%1.3f\t%1.3f\t\t%i\t\t\t%1.6f',xinit, x, iter,
initT);
    disp(row);
end

disp('|')
disp('|')
```

```
disp('Simulated Annealing Hill Climbing');
info = sprintf('First X\tX Result\titerations\tDegradation'); %Common
Column Headers
disp(info);
for i = 0:0.02:0.98
    tic;
    [x, xinit, iter] = HILLCLIMBannealing(i);
    row = sprintf('%1.3f\t%1.3f\t\t%i\t\t\t%1.3f',xinit, x, iter, i);
    disp(row);
end

function [x, xinit, iter] = HILLCLIMBsimple(maxit, g)
    x=rand();
    xinit = x;
    y = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
    t=1;
    countimprove=0;
    notimprove = 1;
    tic;
    while t<maxit && notimprove && ne(x,g) ;%&& x < 0.09 && x > 0.11 x
        x1=x+rand/20;
        y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;

        if y1>y;
            x = x1;
        else
            x1 = x-rand/20;
            y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;
            if y1>y;
                x = x1;
            end
        end
        x;

        improve = (y1-y)/y; improve=abs(improve);
        if improve <= 0.52%improvment margin
            countimprove = countimprove + 1;
        elseif improve <= 0.002 && countimprove >0
            notimprove = 0; %it has improved!!!
        else
            countimprove = 0;
        end
        y=y1;
        t = t+1;
    end
    iter = t;
end

%%Iterated Hill Climb
function [x, xinit, iter]  = HILLCLIMBiterated(n_start, max_it, g)
    x=rand();
    xinit = x;
    y3 = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
    t1=1;
    countimprove=0;
```

```
    notimprove = 1;


    while t1<n_start && ne(x,g)
        x1=rand();
        y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;

        xs=HILLCLIMBsimple(max_it, g);
        y2= 2^(-2*(((xs-0.1)/0.9)^2))*(sin(5*pi*xs))^6;
        t1=t1+1;
        if y2>y1 && y2>y3
            y3 = y2;
            x = xs;
        end
    end %while
    iter = t1;
end% procedure

%%stochastic hill-climbing(max_it, g)
function [x, xinit, iter, initT] = HILLCLIMBstochastic(T, maxit, g)
    x=rand();
    xinit = x;
    y = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
    t=1;
    countimprove=0;
    notimprove = 1;
    initT = 0.1/T;
    while t<maxit && ne(x,g)% && notimprove;
        %y = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
        x1=x+rand/5;
        y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;

        if y1<y
            x1=x-rand/5;
            y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;
        end

        P = (1/(1+exp((y-y1)/initT))); %(1/(1+exp((eval(x)-
eval(x'))/T))) change T [std])
        rP=rand();

        if rP<P
            x = x1;
            y = y1;
        end
        t = t+1;
    end
    iter = t;
end

%%simulated Annealing
function [x, xinit, iter] = HILLCLIMBannealing(G)
    %T = rand()/1000;
    T = 0.25;
    x=rand();
```

```
    xinit = x;
    y = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
    t = 1;
    nostop = 1;
    countimprove = 0;
    while nostop && ne(x, 0.1) && x < 1 && x > 0;
        nostop;
        %STEP 2
        while T >= 0.000001
            for k = 1:4
                perturb = rand/100;
                x1 = x +perturb;
                y = 2^(-2*(((x-0.1)/0.9)^2))*(sin(5*pi*x))^6;
                y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;
                if y1<y
                    x1 = x-perturb;
                    y1 = 2^(-2*(((x1-0.1)/0.9)^2))*(sin(5*pi*x1))^6;
                end
                x1;
                P= ( -abs(exp(y-y1)) / T);
                rP= rand/10;
                if rP > P
                    x = x1;
                    countimprove = countimprove +1;
                end%if
            end

            if countimprove > 3
                nostop = 0;
            end
            T = G*T;
            t = t+1;
        end
    end%while
    iter = t;
end %procedure
```